Tópicos Especiais em Computação II

Trabalho Esquenta 2

Francilândio Lima Serafim (472644)

UNIVERSIDADE
FEDERAL DO CEARÁ

04 de Setembro de 2023

Conteúdo

1 Introdução

No relatório apresentado serão descritas as observações feitas ao executar dois algoritmos que encontram o máximo valor em um array, a primeira versão é baseada em um modelo iterativo e a segunda em um modelo recursivo, sendo suas respectivas complexidades $O(n)$ e $O(n \log n)$. Ambos os algoritmos foram executados recebendo como entradas vetores com valores não ordenados com as seguintes quantidades de valores: 100, 200, 1000, 2000, 5000, 10000, 50000, 100000, 500000, 1000000, 5000000, 10000000 e 100000000.

A comparação dos algoritmos teve como métricas o tempo de execução e consumo de memória para cada entrada supracitada.

2 Metodologia

Os códigos usados para implementar os algoritmos foram escritos na linguagem Python (3.10.6) e executados localmente usando a IDE VS Code.

O código consiste em funções que automatizam as execuções de cada algoritmo sobre as bases através de laços de repetição, após cada execução os dados de consumo de memória, usando a biblioteca *tracemalloc*, e tempo, usando a biblioteca *time*, foram calculados e colocados em um dicionário para posteriormente serem gerados gráficos e tabelas comparativos visando a facilitação de visualizar e entender todo o processo.

Um exemplo de código é mostrado na Figura 1, nele estão as funções de busca do valor máximo versão 1 e a responsável pela automatização das iterações usando as diferentes bases de dados.

```python
def maxVal1(A, n):
    max_val = A[0]
    for i in range(1, n):
        if A[i] > max_val:
            max_val = A[i]
    return max_val

def buscamaxVal1Auto(naoordenados):
  buscamaxval1 = {"memoria":{}, "tempo":{}}

  for arquivo in (naoordenados):
    array = []
    with open(arquivo, 'r') as file:
        reader = csv.reader(file)
        for line in reader:
            for token in line:
                array.append(int(token))

    startTime = time.time()
    tracemalloc.start()
    maxVal1(array, len(array))
    _, totalMemory = tracemalloc.get_traced_memory()
    endTime = time.time()
    tracemalloc.stop()

    totalTime = endTime - startTime

    buscamaxval1['memoria'][arquivo] = totalMemory
    buscamaxval1['tempo'][arquivo] = totalTime
  return buscamaxval1

dicmaxVal1 = buscamaxVal1Auto(naoordenados)

with open("buscamaxV1.pkl", "wb") as f:
    # Usa a função dump para salvar o dicionário no arquivo
    pickle.dump(dicmaxVal1, f)
```

Figura 1: Funções em Python para busca do máximo valor (V1)

3 Resultados

Por convenção, os tamanhos dos dados foram enumerados da seguinte forma:

1. 100
2. 200
3. 1000
4. 2000
5. 5000
6. 10000
7. 50000
8. 100000
9. 500000
10. 1000000
11. 5000000
12. 10000000
13. 100000000

A convenção foi aderida para enxugar o gráfico e tabela utilizados para ilustrar as medições, sendo que nos gráficos os números correspondentes às quantidades de números das bases são colocados no eixo x.

Na Tabela 1 são mostrados os valores em Bytes da memória consumida em cada execução dos dois algoritmos estudados. A partir da análise desses dados é possível concluir que na grande maioria das vezes a memória consumida pelo algoitmo da versão 2 para uma dada entrada é maior que a consumida pelo algoritmo da versão 1 recebendo a mesma entrada, isso pode ser explicado pelo fato de que a versão 2 é recursiva e para achar o valor máximo o vetor de entrada é dividido em partes menores e cada divisão é colocada em uma pilha, um procedimento que requer uma quantidade de memória alocada maior que a necessária para a execução do algoritmo iterativo. É possivel notar que em apenas uma das medições (primeira entrada), a memória consumida pelo algoritmo da versão 1 supera a memória consumida pelo da versão 2, mas isso está em desacordo com o esperado dada a explicação redigida anteriormente, então é possível que se trate de um erro de medição.

Tabela 1: Comparativo de Memória(B) consumida por cada algoritmo

Dados	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
MaxV1	1032779	80	140	140	140	140	140	140	140	140	1016	1016	2720
MaxV2	3240	504	1621	1092	1632	1232	1828	1428	1996	2172	3888	3304	6920

Da Tabela 2 observa-se que os tempos de execução do algoritmo da versão 2 são todos maiores que os da versão 1, o que é esperado pelo fato de que aquele possui uma complexidade de $O(n \log n)$ que

é mais custosa que a complexidade deste, sendo $O(n)$. Ademais, como era esperado conhecendo as complexidades de cada algoritmo, a diferença entre os tempos de execução começa a ser mais visível para as entradas maiores, ou seja, a partir do tamanho 1000000.

Tabela 2: Comparativo de tempo(s) gasto na execução de cada algoritmo

	Tempo(s)	
Dados	**MaxV1**	**MaxV2**
1	0.0	0.001001596450805664
2	0.0	0.0009999275207519531
3	0.0009999275207519531	0.00600886344909668
4	0.0019974708557128906	0.012997865676879883
5	0.003997087478637695	0.033998727798461914
6	0.011000871658325195	0.09600162506103516
7	0.05499720573425293	0.3770005702972412
8	0.10800051689147949	1.1769983768463135
9	0.8719966411590576	4.019001007080078
10	1.1660001277923584	9.655999422073364
11	6.762999534606934	36.16699838638306
12	11.799999713897705	72.05600118637085
13	199.8225758075714	1001.5463080406189

A seguir, para facilitar a visualização da diferença dos comportamentos e possibilitar uma melhor análise da eficiência de cada um serão plotados gráficos. Na Figura 2 é mostrado o gráfico que relaciona as entradas e os gastos de memória na execução de cada algoritmo. Ressalta-se que devido ao comportamento atípico mostrado na primeira entrada (e por prejudicar a análise do gráfico), a mesma foi ignorada na visualização. Logo só são mostradas as medições a partir da segunda entrada (tamanho 200) e pode-se verificar que o gráfico correspondente ao algoritmo V2 está acima do correspondente ao algoritmo V1, evidenciando que a abordagem recursiva é menos eficiente em termos de memória quando comparada à abordagem iterativa.

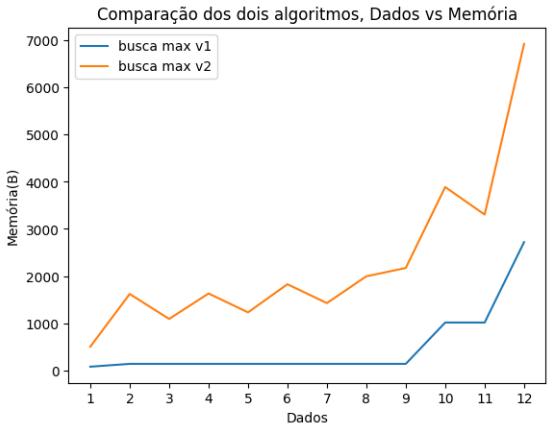

Figura 2: Comparativo da memória

Na Figura 3 é mostrado o gráfico que relaciona as entradas de dados e o tempo de execução de cada

algoritmo, e como supracitado a diferença só começa a saltar aos olhos a partir da décima entrada, por isso não é possível notar muito a diferença na eficiência de cada um olhando as primeiras ocorrências, porém nas últimas fica visível a grande demora para o término da execução da abordagem recursiva em relação à iterativa, com destaque na última entrada para a qual a execução da versão 2 executa com mais de 800 segundos de atraso com relação à versão 1.

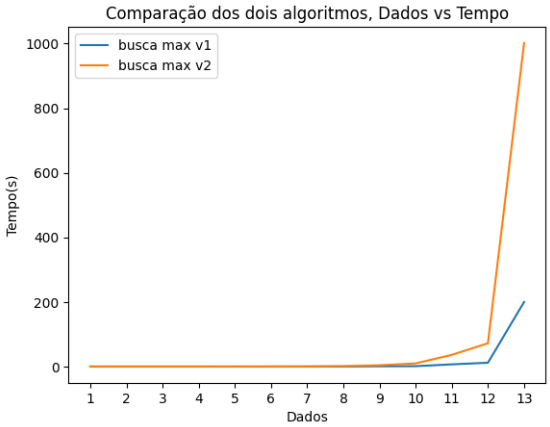

Figura 3: Comparativo do tempo

4 Conclusão

A partir das análises discutidas na seção anterior, pode-se concluir que para a tarefa de achar o valor máximo valor em um array, independente do tamanho da entrada e considerando esta desordenada (pois mesmo sendo esperado igual comportamento no caso ordenado, essa categoria não foi abordada), o algoritmo iterativo é mais eficiente que o recursivo tanto em termos de memória quanto em tempo de execução. Dessa forma, com mínimas exceções a experimentação obteve resultados acordantes com o que se esperava sendo conhecidas as complexidades dos algoritmos.